Trajectory-Based Computation:
Binary Logic, Ket Evolution, and Finite Symbolic Map Engines

Kevin R. Haylett
Manchester, UK
Selected Communications

May 2026

Contents

Chapter 1

Trajectory-Based Computation: Binary Logic, Ket Evolution, and Finite Symbolic Map Engines

Abstract

This chapter develops a finite symbolic and nonlinear-dynamical language for comparing three computational trajectories: conventional binary computation, quantum ket computation, and Finite Symbolic Mechanics / Alphonic map computation. Binary computation is treated as explicit symbolic transformation through discrete states, registers, memory, and logic operations. Quantum computation is reframed as trajectory pre-loading followed by synchronous coupled evolution under Hilbert-space transformation rules and final symbolic flattening through measurement. FSM / Alphonic computation is proposed as a map-based alternative in which relational structure is preloaded into finite symbolic arrays and activated through stable transformations, potentially using optical, CCD, CMOS, or coupled-light systems as fast map-producing and map-detecting coupling layers. The central claim is that computation may be understood more generally as the controlled unfolding of functional symbolic trajectories, where different computational systems differ not only in hardware, but in how they store, transform, compress, stabilise, and measure symbolic relations.

1.1 Introduction: Computation as Trajectory

A computation is often described as an operation on numbers. This description is useful, but it is also heavily compressed. It hides the symbolic preparation that makes the operation possible, the representational constraints that define what a number is allowed to be, the physical substrate that carries the symbolic distinction, and the measurement process by which the result is made available. In ordinary language, we say that a computer "adds two numbers," or that a quantum computer "uses qubits," or that an optical computer "computes with light." Each phrase is functional, but each phrase also conceals a longer symbolic trajectory.

Finite Symbolic Mechanics begins by slowing this language down. A number is not merely a number. A bit is not merely a bit. A qubit is not simply a quantum object. A CCD image is not merely a picture. Each is a finite symbolic structure held within a larger symbolic, physical, historical, and measurement-facing trajectory. A computation, in this framing, is not only a change of state. It is a controlled unfolding of symbolic relations.

We may therefore begin with the following broad form:

$$M_0 \to T_1(M_0) \to T_2(T_1(M_0)) \to M_{\text{out}}, \tag{1.1}$$

where M_0 is an initial symbolic configuration or map, T_1 and T_2 are transformations, and M_{out} is the resulting output configuration. This expression is deliberately broad. It may describe a

binary register being transformed by logic gates, a quantum state being transformed by unitary operators, or a finite symbolic map being transformed into another finite symbolic map.

The chapter compares three computational grammars. The first is conventional binary computation. Here symbols are carried by binary distinctions, usually represented as 0 and 1, and transformed through explicit logic, arithmetic, memory addressing, shifting, branching, and storage. The second is quantum ket computation. In the standard circuit model, quantum computation is described using quantum states, gates, circuits, and measurements [10, 1, 2]. In this chapter, the ket is not treated as a direct image of reality, but as a compressed symbolic directory for a coupled experimental-computational trajectory. The third is FSM / Alphonic map computation. This is the new proposal explored here. In this approach, computation is not primarily performed by sequential binary operations or by Hilbert-space ket evolution, but by preloading stable symbolic relations into finite maps. A transformation then activates these relations, and a detection layer reads an output map. The system may behave partly like a lookup machine, but not merely as a conventional table. It is a relational map engine in which many functional symbolic trajectories may be unfolded in parallel.

The central hinge of this chapter is simple:

$$\text{Computation is not only symbol manipulation.} \tag{1.2}$$

It is also:

$$\text{symbol preparation} \rightarrow \text{trajectory constraint} \rightarrow \text{relational transformation} \rightarrow \text{measurement/readout} \rightarrow \text{decoded} \tag{1.3}$$

This shift matters because it allows binary computers, quantum computers, optical systems, CCD arrays, lookup engines, and Alphonic map engines to be compared without forcing them into the same symbolic basin too early. Each may compute, but each computes by holding and unfolding symbolic function differently.

1.2 Finite Symbols and Functional Symbolic Trajectories

A finite symbol is a bounded, distinguishable symbolic unit available inside the endogenous symbolic space. Examples include:

$$0, \quad 1, \quad |0\rangle, \quad |1\rangle, \quad a_{ij}, \quad b_{ij}, \quad c_{ij}, \quad A, \quad B, \quad C. \tag{1.4}$$

A finite symbol is not assumed to correspond directly to an exogenous thing. It may have been produced through measurement, through convention, through symbolic construction, through apparatus calibration, or through computational encoding. Once it exists in the endogenous symbolic space, it is held by other symbols. It becomes part of a wider symbolic flow.

A functional symbolic trajectory is a finite symbolic pathway that carries a useful relation, prediction, operation, transformation, or measurement-facing function. A binary addition circuit is a functional symbolic trajectory. A quantum circuit is a functional symbolic trajectory. An image transformation from A and B to C is a functional symbolic trajectory. A paragraph

defining a concept is also a functional symbolic trajectory.

The important point is that many different symbolic trajectories may carry the same function. The same arithmetic result may be obtained by binary addition, lookup table, analogue circuit, optical transform, neural network approximation, or quantum subroutine. The function is not uniquely bound to one symbolic pathway.

This prevents premature correspondence. If a quantum formalism works, that does not mean the formalism is uniquely identical to the exogenous system. If a binary algorithm works, that does not mean binary logic is the natural form of the relation being computed. If an image-based map engine works, that does not mean the map is "the thing itself." In each case, the system has stabilised a functional symbolic trajectory.

FSM therefore does not ask first, "what thing is this?" It asks: what symbolic trajectory is being used? What function does it carry? Where was the symbol produced? What has been compressed? What has been flattened? What is being measured? What uncertainty remains?

This language is especially useful when approaching quantum computation, because quantum language often moves quickly from symbols to things. A ket becomes "the state." A wavefunction becomes "the particle." A measurement result becomes "what happened." FSM slows this down. The ket may be useful, extraordinarily useful, but within FSM it remains a finite symbolic structure embedded in a larger functional trajectory.

1.3 The Generonic Boundary as Coupling Layer

In earlier FSM language, the Generonic Boundary can be described as the process by which exogenous interaction becomes finite symbol:

$$\text{Exogenous interaction} \rightarrow \text{Generonic boundary} \rightarrow \text{finite symbol}. \tag{1.5}$$

This is the measurement-facing direction. Something outside the endogenous symbolic space perturbs the measurement system, and a finite symbol appears inside the symbolic system. A detector clicks. A pixel changes value. A voltage crosses a threshold. A bit is written. A result enters the laboratory notebook.

For computation, this one-way description is incomplete. Symbols do not merely receive measurements. Symbols also guide actions. They define experiments. They determine what apparatus is built, what source is activated, what detector is placed, what voltage is applied, what gate is implemented, what pulse is shaped, what threshold is selected, and what result counts as valid.

So the Generonic Boundary is better described here as a coupling layer:

$$\text{Endogenous symbols} \leftrightarrow \text{Generonic coupling layer} \leftrightarrow \text{Exogenous interaction}. \tag{1.6}$$

This is not a claim that symbols magically alter the exogenous world. It is the simpler and more practical claim that symbols guide physical preparation, and physical preparation constrains later symbolic measurement. In computation, this coupling becomes explicit. A program drives hardware. Hardware returns symbols. The returned symbols alter the next

program state.

This also clarifies the role of measurement. A measurement is not merely "data added to theory." A measurement enters the endogenous space as an unstable finite symbol. It may sit at a saddle region of the existing symbolic flow. It may be absorbed by the current model, rejected as error, averaged away as noise, renamed, stabilised, or used to open a new trajectory.

A useful form is:

$$\text{Measurement} \to R_1, \tag{1.7}$$

where R_1 is the first measured result. But R_1 rarely stands alone. Typically, an initial result leads to, or points toward, a following result:

$$R_1 \to R_2. \tag{1.8}$$

However, once R_1 enters the endogenous symbolic space, it is immediately held by all other symbols in that space:

$$R_1 \mid \mathcal{S}, \tag{1.9}$$

where $\mathcal{S}$ is the surrounding symbolic system: prior theory, apparatus history, calibration practice, language, mathematical expectation, uncertainty, and possible future trajectories.

This gives an important FSM statement: a measurement enters FSM as an unstable finite symbol at a saddle region of the existing symbolic trajectory. Typically, an initial result leads to, or points toward, a following result. Once inside the endogenous space, however, the symbol is held by prior symbols, possible future symbols, apparatus history, interpretive language, and the wider symbolic flow.

This matters for computation because every computational result is also a measured symbol. A binary output, a quantum measurement, a CCD pixel array, or an Alphonic output map all become meaningful only after crossing the coupling layer into the endogenous symbolic system.

1.4 Binary Computation as Explicit Symbolic Transformation

Conventional binary computation is the most familiar computational grammar. It stores and manipulates finite symbolic distinctions represented as:

$$0, \quad 1. \tag{1.10}$$

A binary register may be written as:

$$B = (b_1, b_2, b_3, \ldots, b_n), \tag{1.11}$$

where each b_i is a binary symbol and n is the number of binary positions in the register.

A logic operation transforms one symbolic configuration into another:

$$B_0 \xrightarrow{L} B_1, \tag{1.12}$$

where L is a logical, arithmetic, memory, or control operation. A sequence of operations becomes:

$$B_0 \xrightarrow{L_1} B_1 \xrightarrow{L_2} B_2 \xrightarrow{L_3} \cdots \xrightarrow{L_k} B_k. \tag{1.13}$$

Binary computation has enormous strengths. It is general, robust, programmable, scalable, and deeply engineered. It supports memory, branching, iteration, modularity, abstraction, error detection, error correction, and symbolic layering. Modern computing civilisation is built from this grammar.

Yet binary computation also carries a cost. Many relations must be computed explicitly each time they are required. Even when a relation is stable, repeatable, and known in advance, a binary system may still need to derive it through sequential or parallel logic operations unless the relation has been stored.

A lookup table shifts this balance. Instead of recomputing:

$$f(a, b), \tag{1.14}$$

the system stores the result and retrieves it:

$$(a, b) \to f(a, b). \tag{1.15}$$

This already hints at the FSM / Alphonic direction. A lookup table is not less computational because it retrieves rather than derives. It has simply moved the cost. The cost was paid during table construction, storage, addressing, and validation. The live computational act becomes selection and retrieval.

This gives a first useful distinction: binary computation often computes relations each time they are needed; lookup-based computation stores prior relational structure and activates it when required. The Alphonic map idea extends this logic, but replaces the single lookup table with a stable symbolic map structure capable of holding many relations in parallel.

1.5 Quantum Ket Computation in Standard Language

Quantum computation is usually formulated in the language of states, gates, circuits, and measurements. In the circuit model, a system is prepared, transformed by quantum gates, and measured [10, 1, 2].

For a single qubit, a pure state is written:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.16}$$

where α and β are complex amplitudes and:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{1.17}$$

For n qubits, the state is written:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle, \tag{1.18}$$

where $c_i \in \mathbb{C}$, $|i\rangle$ are computational basis states, and:

$$\sum_i |c_i|^2 = 1. \tag{1.19}$$

A quantum circuit applies transformations to this state. In the simplest pure-state form:

$$|\psi_0\rangle \to U_1|\psi_0\rangle \to U_2U_1|\psi_0\rangle \to \cdots \to |\psi_f\rangle \to \text{measurement}. \tag{1.20}$$

Here $U_1, U_2, \ldots$ are unitary operators representing quantum gates. In practice, quantum computing also involves density matrices, channels, noise, decoherence, hardware constraints, compilation, calibration, and measurement. State preparation itself is a substantial issue: the preparation of useful quantum states is an active topic because quantum algorithms depend on how states are initialised, encoded, and measured [11, 9].

The standard language is operationally powerful. It predicts measurable distributions. It provides a formal grammar for designing algorithms. It enables simulation, compilation, circuit optimisation, and hardware control.

But FSM asks us not to skip the symbolic layers. The ket is not the experimental system itself. It is a compressed symbolic representation used to coordinate preparation, transformation, prediction, and measurement. The qubit is not merely a small "thing." It is a flattened symbolic geometry. It stands in for a long experimental-symbolic trajectory.

A physical qubit may involve superconducting circuits, trapped ions, photons, spins, defects, or other systems. But in the computational formalism, this richness is compressed into $|0\rangle$, $|1\rangle$, amplitudes, gates, and measurements. The qubit appears simple because much of the geometry has already been folded away.

1.6 Quantum Computation Reframed in Nonlinear-Dynamical Language

Finite Symbolic Mechanics reframes quantum computation using nonlinear-dynamical language without denying the standard formalism.

In standard quantum-computing language:

$$\text{state preparation} \to \text{unitary evolution} \to \text{measurement} \to \text{classical output}. \tag{1.21}$$

In FSM / NLD language:

$$\text{trajectory pre-loading} \to \text{synchronous coupled trajectory evolution} \to \text{relational interference} \to \text{symbolic flatte} \tag{1.22}$$

The correspondences are as follows. State preparation becomes *trajectory pre-loading*. Unitary

evolution becomes *coupled trajectory evolution.* Entanglement becomes *non-separable symbolic coupling.* Measurement becomes *symbolic flattening.* The ket becomes a *compressed symbolic trajectory directory.*

This reframing matters because it avoids the misleading popular phrase that a quantum computer "tries all answers at once." The more careful statement is that a prepared coupled state evolves as a whole under transformation rules, and the final measured distribution is shaped by the structure of that evolution.

The quantum computer is therefore not simply a parallel binary machine. It is not performing many ordinary classical calculations side by side. It is an engineered physical-symbolic system in which a coupled trajectory evolves before being flattened into measurable symbols.

In FSM notation:

$$\text{Quantum advantage} \sim \text{useful transformation of a preloaded coupled trajectory before flattening.} \tag{1.23}$$

The word *preloaded* is important. A quantum computer does not begin from nothing. It begins from a prepared state, an encoding, an ansatz, an input distribution, a circuit design, and a measurement scheme. The later evolution is constrained by this preparation. In FSM language, the initial ket-state establishes the symbolic-physical conditions under which the later coupled evolution can unfold.

This brings quantum computing closer to other forms of computation. It has a different grammar, but not a magical exemption from preparation, representation, coupling, measurement, and error correction.

1.7 The Qubit as Flattened Geometry

A single qubit may be represented using the Bloch sphere:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \tag{1.24}$$

where θ and ϕ define a point on the Bloch sphere. This partially reintroduces geometry. But even the Bloch sphere is already a symbolic geometry. It does not restore the full exogenous physical system.

The qubit representation has compressed many layers:

$$\text{Exogenous system} \to \text{apparatus coupling} \to \text{signal formation} \to \text{calibration} \to \text{thresholding} \to \text{binary outcome} \tag{1.25}$$

This is why FSM must be careful not to assume direct correspondence. The qubit formalism works. It is not rejected. But it is held inside a wider symbolic dynamical system.

The relationship may be written:

$$\text{QM} \sim f(\text{FSM}), \tag{1.26}$$

meaning that quantum mechanics may be treated as a functionally stabilised symbolic subsystem

within FSM. More explicitly:

$$\mathrm{QM} \in \mathcal{T}_{\mathrm{FSM}}, \tag{1.27}$$

where $\mathcal{T}_{\mathrm{FSM}}$ is the space of finite functional symbolic trajectories.

Within FSM, this is inescapable. Once quantum mechanics is expressed using finite symbols, equations, kets, operators, diagrams, probabilities, measurements, apparatus descriptions, and interpretive language, it is already inside the endogenous symbolic flow. QM does not stand outside FSM; rather, it becomes one highly successful trajectory within the wider nonlinear-dynamical symbolic space.

This is not a demotion of quantum mechanics. It is a clarification of its symbolic status.

1.8 Stable Images and Parallel Symbolic Arrays

We now shift from ket computation to map computation.

Suppose we have a stable image A, a stable image B, and a resulting stable image C, generated by an experimental setup:

$$(A, B) \to C. \tag{1.28}$$

The images may be arbitrary. They may be light patterns, sensor maps, projected symbols, numeric encodings, spatial masks, CCD frames, or other finite symbolic arrays. What matters is not that they are pictures in the ordinary sense. What matters is that they are stable symbolic maps.

Let:

$$A = \{a_{ij}\}, \tag{1.29}$$

$$B = \{b_{ij}\}, \tag{1.30}$$

$$C = \{c_{ij}\}, \tag{1.31}$$

where i and j are finite spatial indices.

In an ordinary image-processing frame, C might be treated as an output picture. In FSM, C is an output symbolic relation field. Each output element may be treated as:

$$c_{ij} = f_{ij}(A, B), \tag{1.32}$$

where f_{ij} is the local or nonlocal function that maps the input maps A and B into the output element c_{ij}.

The family of output functions is:

$$\mathcal{F}_{AB\to C} = \{f_{ij}\}. \tag{1.33}$$

This is the key shift. The output map is not merely a grid of values. It is an array of functional symbolic relations.

A CCD or CMOS detector is already a massively parallel measurement surface. Each pixel can integrate light during an exposure and return a finite symbol. Optical computing, photonic computing, and optical information processing have long explored the use of light and optical

systems for computation and information transformation [3, 13, 17]. Diffractive optical neural networks are another related field, using diffractive layers and optical propagation to perform learned transformations [7].

However, the FSM / Alphonic proposal is not simply "optical computing." It shifts the emphasis.

Existing optical computing often says:

$$\text{light performs computation.} \tag{1.34}$$

The FSM map proposal says:

$$\text{light creates or activates maps,} \tag{1.35}$$

$$\text{the maps hold the computational compression,} \tag{1.36}$$

$$\text{the output is a parallel symbolic relation field.} \tag{1.37}$$

This distinction matters. The optics may be the coupling layer. The CCD may be the readout layer. But the computational leverage lies in the maps and their preloaded relational structure.

1.9 The Alphonic Map Engine

The Alphonic map engine is a proposed computational grammar that does not begin from binary shift registers. It begins from finite symbolic maps and stable map transformations.

The basic form is:

$$(A, B) \xrightarrow{T} C, \tag{1.38}$$

where A and B are input maps, T is a stable transformation or coupling operation, and C is an output map.

The output map may contain many preconfigured relations at once. If A holds a number a, and B holds a number b, then different regions or layers of C could hold:

$$a + b, \tag{1.39}$$

$$a - b, \tag{1.40}$$

$$ab, \tag{1.41}$$

$$\frac{a}{b}, \tag{1.42}$$

$$a > b, \tag{1.43}$$

$$a = b, \tag{1.44}$$

$$a^2 + b^2, \tag{1.45}$$

or other functions chosen by the map design.

This resembles lookup, but it is not merely a conventional lookup table. It is a light-triggered relational lookup array, or more broadly, a trajectory-based map computation.

The "program" is partly in the predefined maps and partly in the stable transformation layer. Once the maps are defined and the system is calibrated, the activation event unfolds many relational outputs in parallel.

A useful expression is:

$$(A, B, \mathcal{F}) \xrightarrow{T} C, \tag{1.46}$$

where $\mathcal{F}$ is the preloaded family of relations available to the transformation. The computation is not performed by deriving every relation step by step at runtime. The relational structure has already been placed into the maps. Runtime becomes activation, detection, and decoding.

This gives the central practical claim: the computational function is held in the predefined map relations. The physical system supplies fast activation and detection. The practical question is whether the cost of defining, storing, stabilising, and decoding the maps is lower than computing the same relation through binary operations.

This is a trade-off, not a miracle. The Alphonic engine does not automatically replace binary computation. It may be advantageous where the cost of map preparation and detection is lower than repeated binary derivation for a useful class of tasks.

1.10 The Three Computational Approaches

We can now compare the three computational grammars.

In binary computation, the symbolic unit is the bit:

$$0, \quad 1. \tag{1.47}$$

The storage medium is the register, memory cell, address space, or binary array. The transformations are logic gates, arithmetic units, shifts, masks, comparisons, branches, and memory operations. The strength of the system is generality, reliability, programmability, and mature engineering. The weakness is that relations often have to be computed explicitly, even when they are stable and repeatedly used.

In FSM language, binary computation is:

$$\text{sequential or parallel binary symbolic trajectory.} \tag{1.48}$$

In quantum ket computation, the symbolic unit is the ket:

$$|\psi\rangle, \quad |0\rangle, \quad |1\rangle, \quad \sum_i c_i |i\rangle. \tag{1.49}$$

The storage medium is the prepared quantum state, held through a physical system and represented through Hilbert-space formalism. The transformations are unitary gates, quantum channels, controlled operations, measurement procedures, and error-correction structures. The strength is the ability to transform a coupled state representation before measurement. The weakness is fragility, noise, preparation cost, readout cost, and heavy error-correction overhead

[12].

In FSM language, quantum computation is:

$$\text{preloaded coupled ket trajectory.} \tag{1.50}$$

In FSM / Alphonic map computation, the symbolic unit is the finite symbolic map:

$$A, \quad B, \quad C, \quad \mathcal{F}_{AB\to C}. \tag{1.51}$$

The storage medium is the map relation itself. The transformation is:

$$(A, B) \xrightarrow{T} C. \tag{1.52}$$

The strength is that many preconfigured relations may be activated and detected in parallel. The weakness is map definition cost, calibration, error correction, resolution limits, detector noise, source drift, decoding complexity, and possibly limited generality unless the maps are richly reconfigurable.

In FSM language, Alphonic map computation is:

$$\text{trajectory-based relational map computation.} \tag{1.53}$$

Compressed into one comparison:

$$\text{Binary computation: compute the relation.} \tag{1.54}$$

$$\text{Quantum computation: evolve the coupled trajectory.} \tag{1.55}$$

$$\text{FSM map computation: activate the preloaded relation field.} \tag{1.56}$$

This is not a hierarchy. It is a comparison of symbolic grammars.

1.11 Relation to Reservoir Computing and Optical Computing

The FSM map proposal sits near several existing fields, but it does not collapse neatly into them.

Reservoir computing is especially relevant. Reservoir computing uses a dynamical system, called the reservoir, whose internal state is driven by input; a readout layer is then trained to extract the desired output [4, 8]. Photonic reservoir computing has also been developed as a hardware route, with photonic platforms used as physical reservoirs [6, 16].

This is close to the FSM direction, because both approaches exploit a physical or symbolic dynamical system rather than imposing all computation through explicit binary logic. Reservoir computing says: let the system's dynamics expand the input into a rich state, then train a readout.

FSM map computation says something adjacent but not identical: define stable symbolic maps so that the relation field itself carries preloaded computational structure. The physical transformation activates a map relation rather than acting as the whole computational

explanation.

Diffractive optical neural networks are also close. They use optical propagation through diffractive structures to implement learned transformations [7]. Again, FSM differs in emphasis. It does not deny that optics may transform the signal. It instead asks where the symbolic compression lives. In many optical systems, the optical path, mask, lens, diffractive layer, source, and detector jointly create the transformation. FSM asks whether the maps themselves can be treated as the main computational substrate.

So the FSM formulation is:

$$\text{optics} = \text{map-producing coupling layer}, \tag{1.57}$$

not necessarily:

$$\text{optics} = \text{the whole computer}. \tag{1.58}$$

This is a subtle but important distinction. The optics create, select, combine, or activate maps. The computation is read in the stable relation between maps.

1.12 Stable Source Systems and Near-Field Map Formation

The near-field light-source experiment offers a useful conceptual bridge.

Suppose a single LED or light source produces a stable pattern over time. The first few detections are not merely isolated data points. They are early samples of a stable unfolding trajectory:

$$D_1 \to D_2 \to D_3 \to \cdots \to P, \tag{1.59}$$

where D_1, D_2, D_3 are early detections and P is the later stabilised pattern.

If the system is stable, the early detections already point toward the later full pattern. They do not predict it magically. Rather, they are finite samples of an unfolding stable trajectory.

Now consider two nearby sources:

$$S_1 \to P_1, \tag{1.60}$$

$$S_2 \to P_2. \tag{1.61}$$

If both sources are activated together:

$$S_1 + S_2 \to P_{12}. \tag{1.62}$$

The central comparison becomes:

$$\Delta P = P_{12} - (P_1 + P_2). \tag{1.63}$$

If:

$$\Delta P \approx 0, \tag{1.64}$$

then the system is mostly additive. If ΔP has stable structure, the combined activation has produced a non-additive map relation.

This is not merely an optical question. It is a symbolic-computational question. If stable source systems can produce stable non-additive maps, then those maps may be usable as computational relation fields. A coupled light-source system, paired with CCD detection, becomes a testbed for map-based computation.

The critical issue is stability. If the array is stable, then the parallel symbolic relations are stable. Each pixel-symbol is not merely an independent number. It is held by neighbouring pixels, optical spread, diffraction, sensor geometry, exposure time, source behaviour, thresholding, and the global pattern. The array becomes:

$$\mathcal{A} = \{p_i,\ r_{ij},\ R_{\text{global}}\}, \tag{1.65}$$

where p_i are pixel-symbols, r_{ij} are inter-element relations, and R_{global} is the stabilised whole-array relation.

The calculation is therefore not only in the pixel values. It is in the stable relational geometry across the array.

1.13 Preloading, Lookup, and the Cost of Computation

Preloading is not unique to FSM map computation. Quantum computers preload through state preparation. Binary computers preload through programs, memory, constants, lookup tables, caches, models, compiled instructions, and hardware design. Optical systems preload through masks, lenses, diffractive layers, spatial light modulators, alignment, and calibration.

The question is not whether preloading occurs. The question is where the cost is paid and where the compression lives.

For binary computation, the cost may be expressed roughly as:

$$C_{\text{binary}} = C_{\text{operation}} \times N_{\text{steps}}, \tag{1.66}$$

where $C_{\text{operation}}$ is the cost of each operation and N_{steps} is the number of operations required.

For FSM map computation:

$$C_{\text{FSM}} = C_{\text{map setup}} + C_{\text{activation}} + C_{\text{detection}} + C_{\text{decoding}} + C_{\text{correction}}. \tag{1.67}$$

The map engine is advantageous when:

$$C_{\text{FSM}} < C_{\text{binary}} \tag{1.68}$$

for a useful class of repeated or parallel tasks.

This is the correct trade-off. The FSM / Alphonic engine does not evade computation. It relocates it. It pays cost in map design, map storage, calibration, stability, and decoding, then seeks advantage through rapid activation and parallel detection.

This may be especially relevant for functions that are stable, repeated, spatially representable, relation-heavy, parallelisable, lookup-friendly, tolerant of finite uncertainty, or naturally expressed as maps. It may be less useful for tasks requiring arbitrary symbolic branching, complex dynamic

control, exact sequential dependency, or frequent reconfiguration.

The honest claim is therefore: FSM map computation may be powerful where preloaded relational structure can be activated and detected faster than the same relation can be derived through binary operations.

1.14 Error Correction and Uncertainty

Any serious computational proposal must address error.

Quantum computing requires error correction because physical quantum systems are noisy and difficult to maintain [12, 14]. FSM map computation will also require correction. Possible error sources include map instability, source drift, detector noise, pixel defects, alignment changes, nonlinear saturation, temperature effects, exposure variation, thresholding error, cross-talk between map regions, decoding ambiguity, and finite resolution.

This is not a fatal objection. It is a design requirement.

An output map may be written not merely as:

$$C = \{c_{ij}\}, \tag{1.69}$$

but as:

$$C = \{(c_{ij}, u_{ij})\}, \tag{1.70}$$

where u_{ij} is local uncertainty associated with the output symbol c_{ij}.

A stable transformation may be defined by repeated outputs remaining within an accepted bound:

$$d(C_k, C_l) < \epsilon, \tag{1.71}$$

where d is a chosen distance measure between output maps, C_k and C_l are outputs from repeated runs, and ϵ is the tolerated uncertainty.

Correction methods could include repeated exposures, calibration maps, redundancy, neighbourhood voting, attractor classification, confidence maps, difference maps, and stable basin decoding.

A difference map is especially important:

$$\Delta C = C_{\text{measured}} - C_{\text{expected}}. \tag{1.72}$$

This allows the system to detect drift, noise, or unexpected coupling structure. In FSM language, error correction is not merely technical cleanup. It is the process by which unstable measured symbols are stabilised within the endogenous symbolic flow.

1.15 Simulation Before Hardware

Before building physical hardware, FSM map computation can be simulated.

Quantum computers are often simulated by representing state vectors or density matrices and applying gate operations. FSM map computation can be simulated in an analogous broad

sense, though not using Hilbert-space kets. The simulator would represent finite symbolic maps and apply map transformations:

$$M_0 \to T_1(M_0) \to T_2(T_1(M_0)) \to M_{\text{out}}. \tag{1.73}$$

Or, for the two-map case:

$$(A, B) \xrightarrow{T} C. \tag{1.74}$$

The simulator would test whether maps can encode useful relation families, how many functions can be held per output map, how noise affects decoding, how lookup speed compares with binary calculation, how compression scales, and what classes of tasks suit map computation.

This is an important practical step because the concept does not require immediate physical implementation. The symbolic map grammar can be developed first. Hardware then becomes a possible instantiation of a previously tested symbolic-computational model.

This also matches the wider FSM principle: do not assume correspondence. A software FSM map simulator is not the physical Alphonic engine. It is a symbolic trajectory used to explore whether the formalism is coherent, stable, and useful.

1.16 Relation to Takens, NLD, and Phase-Space Reconstruction

The nonlinear-dynamical framing is not decorative. It is foundational.

Takens-style embedding ideas are relevant because they show how dynamical structure can be reconstructed from observed signals under suitable conditions [15, 5]. FSM does not simply import Takens' theorem as a finished answer. Rather, it draws a methodological lesson: observed symbolic sequences may carry more of the underlying dynamical structure than they appear to carry when treated as isolated points.

This applies to language, measurement, image formation, quantum readout, and map computation.

A measured symbol is not just a point. It may be a trajectory sample. An image is not just an array. It may be a relational field. A qubit is not just a two-state object. It may be a compressed symbolic directory for a coupled trajectory. A computation is not just an operation. It may be the unfolding of a preloaded trajectory through a constrained symbolic space.

The Alphonic engine idea belongs in this wider NLD language. It treats computation as controlled symbolic flow, not merely as static symbol manipulation.

1.17 What Is New, and What Is Not

It is important to be careful.

The following are not new by themselves: binary computation, lookup tables, optical computing, CCD/CMOS imaging, reservoir computing, photonic computing, diffractive neural networks, quantum circuit simulation, or parallel image processing. These existing fields are real and substantial [3, 4, 7, 16].

The possible novelty lies in the FSM synthesis. The chapter proposes a language in which

binary computation, quantum computation, and map computation can be compared as different ways of holding functional symbolic trajectories.

It also proposes a map-first interpretation of a possible Alphonic engine. The optics create or activate maps. The detector reads maps. The compression is held in the stable relation between maps. The computation is the activation of a preloaded symbolic relation field.

This framing may help avoid two common traps. The first trap is treating quantum computation as magical parallelism. The second trap is treating optical computation as if the optical process alone is the whole explanation. FSM instead asks where the functional symbolic trajectory is held, how it is preloaded, how it is transformed, how it is measured, and how its uncertainty is stabilised.

1.18 Conclusion: Computation as Controlled Symbolic Unfolding

The three computational approaches may now be summarised.

Binary computation transforms discrete symbolic states through explicit logic. Quantum computation preloads a coupled ket trajectory, evolves it under Hilbert-space rules, and flattens the result through measurement. FSM / Alphonic map computation preloads relational structure into finite symbolic maps and activates stable transformations that unfold many symbolic relations in parallel.

The deeper comparison is not hardware alone. It is symbolic geometry.

Binary computation stores function in instructions, registers, memory, and logic. Quantum computation stores function in state preparation, ket representation, coupled evolution, and measurement statistics. FSM map computation stores function in predefined symbolic maps and the relation field between them.

The common structure is:

$$\text{preparation} \rightarrow \text{transformation} \rightarrow \text{measurement} \rightarrow \text{symbolic result}. \tag{1.75}$$

The difference lies in where the compression lives. For binary computation, compression lives in algorithms, instruction sets, memory structures, and logic circuits. For quantum computation, compression lives in ket representation, state preparation, unitary transformation, and measurement design. For FSM map computation, compression lives in stable symbolic maps and their preloaded relation fields.

This gives the final compressed formulation:

$$\text{Binary computation: compute the relation.} \tag{1.76}$$

$$\text{Quantum computation: evolve the coupled trajectory.} \tag{1.77}$$

$$\text{FSM map computation: activate the preloaded relation field.} \tag{1.78}$$

The broader claim is not that one computational grammar must replace the others. The broader claim is that computation itself can be understood as the controlled unfolding of finite

symbolic trajectories. From this viewpoint, a computer is not merely a machine for calculating. It is a device for preparing, constraining, transforming, measuring, and stabilising symbols.

That is the opening for FSM computation: not a rejection of existing computation, not a rejection of quantum computation, not a rejection of optical computation, but a wider symbolic-dynamical container in which each can be held, compared, unfolded, and, where possible, recombined.

Chapter 2

Near-Field Map Formation: Experimental Demonstration

The theoretical development above grew directly from a simple bench-top experiment performed by the author in the near-field regime (source–detector separation shorter than the wavelength of the emitted light). The goal at the time was modest: to look for evidence of internal photon structure using a conventional photon-counting mindset. What emerged instead was the first concrete evidence for the map-based computational grammar developed in this chapter.

2.0.1 Experimental Setup

A standard LED had its plastic lens removed, exposing a compact, flat emitting surface. This surface was placed a few hundred micrometres from the face of a cooled CCD sensor (typical astronomy-grade camera). Astronomy image-stacking software was used to accumulate single detectable events while the LED was driven at extremely low current, so that photons (or whatever underlying process) arrived one detectable event at a time. No lenses, no spatial light modulators, and no deliberate diffractive elements were used; the geometry was deliberately minimal.

The result was striking. After sufficient accumulation the pattern on the CCD formed an extremely clear, high-contrast image of the LED emitting surface. More importantly, the image was *perfectly repeatable* across independent runs. The same source geometry always produced the same final map $P = \{p_{ij}\}$, to within the limits of detector noise and thermal drift.

2.0.2 Stability and Early Detection

The key observation, and the one that redirected the entire theoretical trajectory, was this: the system did not need the full accumulation to identify the final map. After only the first handful of detectable events the partial map already carried enough information to determine, with high confidence, which stable pattern was being realised. In other words, the early detections $D_1, D_2, D_3, \ldots$ were not independent random arrivals. They were finite samples of a *single, stable functional symbolic trajectory* that the entire coupled system had entered.

Formally, a single stable source configuration S produces an output map:

$$S \to P. \tag{2.1}$$

Different source configurations (different LEDs, or different combinations of LEDs placed side-by-side in the near field) produce distinct maps:

$$S_1 \to P_1, \qquad S_2 \to P_2, \qquad S_1 + S_2 \to P_{12}. \tag{2.2}$$

The non-additive structure is captured by the difference map

$$\Delta P = P_{12} - (P_1 + P_2), \tag{2.3}$$

which is itself stable and repeatable. Once a few early events have arrived, the system has already selected one specific stable mode; the remaining events simply fill in the already-determined relational field.

This is not a wave-interference or single-photon narrative. It is a stability narrative. The near-field coupling itself pre-loads a complete symbolic map whose internal relations are fixed by the geometry of the source–detector system. The CCD array acts as the Generonic coupling layer, converting the exogenous interaction into a finite symbolic array P. The first few events do not "build" the image in the ordinary sense; they *point toward* the already-chosen stable trajectory.

2.0.3 From Stable Maps to Relational Fields

The experiment supplies the physical substrate for the Alphonic map engine proposed earlier. Two input maps A and B (the individual stable patterns from each source) are combined through the fixed physical transformation T (near-field overlap). The output map C now contains, in parallel, multiple decoded relations: the sum pattern, the difference pattern, and any other functional relation the experimenter chooses to read out from the same pixel array.

In FSM notation:

$$(A, B) \xrightarrow{T} C, \qquad C = \{c_{ij} = f_{ij}(A, B)\}, \tag{2.4}$$

where the family $\mathcal{F}_{AB\to C} = \{f_{ij}\}$ is pre-loaded by the stable source geometry. The "program" is partly in the physical coupling and partly in the decoding layer. Runtime cost collapses to activation (turn on the sources) plus parallel detection plus lightweight decoding of the partial map. No sequential logic steps are required.

The original photon-structure motivation was not refuted; it simply became irrelevant. The symbols that worked inside the older trajectory ("photon", "wave packet", "interference") were replaced by a more economical grammar: stable source configuration, stable output map, and early samples of a functional symbolic trajectory. The exogenous interaction is the same; only the endogenous symbolic description has changed.

This near-field demonstration shows that trajectory-based map computation is not an analogy. It is already occurring, today, in a room-temperature, table-top system built from commodity components. The physics itself supplies the pre-loading; the detector supplies the parallel

readout; the experimenter supplies the decoding map. The result is a concrete, repeatable instance of

$$\text{trajectory pre-loading} \rightarrow \text{stable relational transformation} \rightarrow \text{parallel symbolic readout}. \quad (2.5)$$

The experiment therefore serves two roles. First, it anchors the abstract claims of Sections 7–11 in direct observation. Second, it supplies the missing concrete example that lets the reader visualise exactly how an Alphonic map engine might operate in practice: define the stable sources once, activate them, read the partial map, and decode multiple pre-loaded functions in parallel. The computational compression lives in the stable map relations themselves. Everything else is activation and detection.

Figure 2.1: Schematic of the near-field LED–CCD setup and the early-detection phenomenon. The partial map after only a few events already correlates strongly with the final stable image.

Bibliography

[1] D. Deutsch, "Quantum theory, the Church–Turing principle and the universal quantum computer," *Proceedings of the Royal Society of London A*, vol. 400, no. 1818, pp. 97–117, 1985.

[2] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467–488, 1982.

[3] J. W. Goodman, *Introduction to Fourier Optics*, 3rd ed. Roberts and Company, 2005.

[4] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," GMD Report 148, German National Research Center for Information Technology, 2001.

[5] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*, 2nd ed. Cambridge University Press, 2004.

[6] L. Larger et al., "High-speed photonic reservoir computing using a time-delay-based architecture: million words per second classification," *Physical Review X*, vol. 7, 011015, 2017.

[7] X. Lin et al., "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018.

[8] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[9] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Transformation of quantum states using uniformly controlled rotations," *Quantum Information & Computation*, vol. 5, no. 6, pp. 467–473, 2005.

[10] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th anniversary ed. Cambridge University Press, 2010.

[11] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Physical Review A*, vol. 83, 032302, 2011.

[12] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[13] D. Psaltis and F. H. Mok, "Holographic memories," *Scientific American*, vol. 273, no. 5, pp. 70–76, 1995. doi:10.1038/scientificamerican1195-70.

[14] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical Review A*, vol. 52, no. 4, R2493–R2496, 1995.

[15] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics, vol. 898, pp. 366–381. Springer, 1981.

[16] G. Tanaka et al., "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[17] D. Woods and T. J. Naughton, "Optical computing: Photonic neural networks," *Nature Physics*, vol. 8, no. 4, pp. 257–259, 2012.